Benchmarking of software for mobile devices

Christian Strømme
Bergen University College
+47 90 19 02 94
chris.stromme@gmail.com

Helge Welde
Bergen University College
+47 90 65 30 54
helgewelde@gmail.com

ABSTRACT
This is a summary of a report we have written as part of our bachelor project called *Benchmarking of software for mobile devices.* Our report aimed to provide the reader access to unbiased documentation regarding which kind of software is the most beneficial for use on a mobile platform.

INTRODUCTION
We developed an example application which mimics basic functionality found in common applications, such as:

- Reading/writing text from/to files.
- Displaying 2D and 3D graphics.
- Loading web pages.

Two versions of the application were made, each version utilizing different graphical frameworks, respectively GTK+ [1] and Qt [2]. The applications were written in C and C++. The concept of the project was to run the applications on a mobile platform, namely the Nokia N810 «internet tablet».

With the finished applications in hand we conducted a series of benchmarking tests on each version; measuring memory and CPU usage. These tests were also conducted on existing applications such as web browsers and word processors.

During the development we published our results on the project web page: http://hib2009.dyndns.org

BENCHMARKING TOOLS
We performed the tests on a single core laptop. Most of the tests were not practical, or in some cases even possible, to run on the N810. A short introduction of the tools we used follows:

Memory usage

Exmap
Exmap is a utility which takes a snapshot of how the physical memory and swap space are currently used by all the processes on your system. It examines which page of memory are shared between which processes, so that it can share the cost of the pages fairly when calculating usage totals [3].

The two most interesting columns in Exmap are *Effective resident-* and *Resident Memory*. The *resident* size is the total amount of memory used by a process, but only in RAM (not swap). The *effective resident* value is equal to the *resident*, but it is adjusted for shared memory. That is: memory that is shared by two or more processes gets divided equally amongst them.

Valgrind
Valgrind [4] is a suite for debugging and profiling memory-related issues in Linux applications. We used two of Valgrinds tools in this project:

- *Memcheck* – detects memory-management problems.
- *Massif* – a heap profiler.

The Valgrind tools were primarily used when we needed to conduct a more detailed memory analysis.

CPU usage
The CPU load a process uses can be measured using a combination of *ps* and *watch*. The *ps* program returns the desired information about a process. *watch* runs the process repeatedly, allowing us to take snapshots of the CPU-usage, and to track its development over time. We wrote a small script for this purpose.

EXISTING APPLICATIONS
The following is a selection of the more interesting results from our tests on existing applications.

PDF readers
Evince [5] and Okular [6] are the default PDF/document readers for Gnome and KDE. They provide a similar level of functionality. ePDFView [7] is a lightweight alternative, offering only a basic level of functionality: a good choice for the mobile platform. All applications use Poppler, a library for rendering PDF.

We conducted this test by loading a PDF version of *Structure and Interpretation of Computer Programs*, by Abelson, Sussman, and Sussman. The document is 634 pages and the binary size is 4.4MB. See Table 1 for summary memory usage.

If we look at the difference between resident and effective memory consumption, Evince uses 3MB less in favour of Okular. The explanation is probably that, in addition to

Application	Effective Resident (KB)	Resident (KB)
ePDFView	16101	21744
Evince	25563	31620
Okular	22596	31564

Table 1: Memory usage, PDF readers

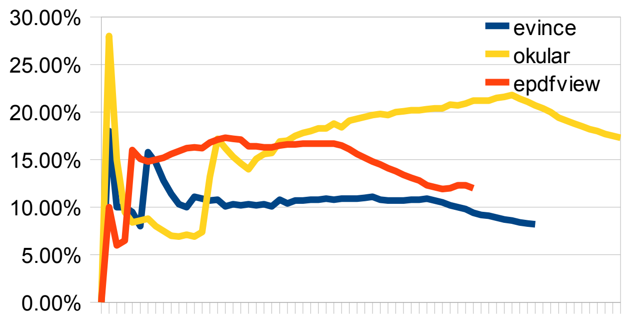

Figure 1: CPU usage, PDF readers

Okular, various shared KDE libraries run in the background (e.g. Klauncher) making the effective resident consumption smaller. If we compare the resident column for both applications, the results correspond better with each other.

If we look at the difference between resident and effective memory consumption, Evince uses 3MB less in favour of Okular. The explanation is probably that, in addition to Okular, various shared KDE libraries run in the background (e.g. Klauncher) making the effective resident consumption smaller. If we compare the resident column for both applications, the results correspond better with each other.

We conducted the CPU test in a similar fashion as the memory test, but this time we scrolled down 50 pages, then let the applications idle for 10 seconds. The results is shown in the graph in Figure 1.

At start-up Okular has the highest peak value, in addition to consuming the most CPU when scrolling down the document. What strikes us as odd, is that the extra CPU-usage does not look like it is being used efficiently. Evince uses the least amount of CPU cycles of the three readers, but still *feels* more responsive throughout the test, followed closely by ePDFView.

Web browsers

We measured memory- and CPU usage on two web browsers: Midori [8] and Arora [9]. They are both relatively feature poor, compared to e.g. Firefox. Both use the *WebKit* web rendering engine.

Midori is written in GTK+. Though it is still in alpha state and suffers from some stability issues, it is an interesting browser. This is the browser intended to be default in future XFCE releases.

Application	Effective Resident (KB)	Resident (KB)
Midori	86915	91264
Arora	112788	115100

Table 2: Memory usage, web browsers

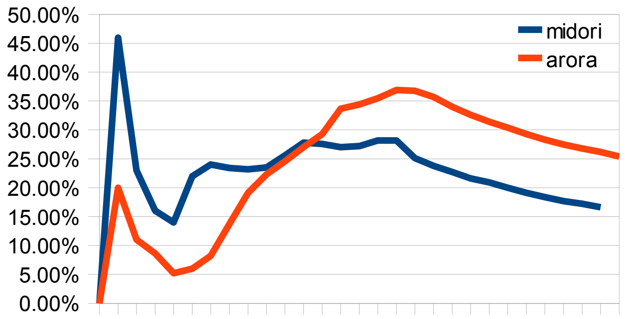

Figure 2: CPU usage, web browsers

Arora is based on Qt and does not have KDE dependencies. In terms of functionality Arora is the minimalist one.

We ran tests on both browsers when displaying: a blank page, one web page, 5 web pages. Table 2 shows the memory usage after loading 5 different web pages.

Arora used less memory initially, but as soon as we loaded more web pages it used slightly more than Midori. We suspect Arora's higher memory usage being caused by:

- libQtWebkit using more memory. The isolated results from this test do not enable us to support this conclusion unconditionally. However, the test results are congruent with the results from the tests performed on our own GTK+ and Qt applications. The result found when profiling the memory allocations with Massif, shows that QImage, or the way QImage is used in libQtWebKit, may be the reason for the elevated memory usage.
- Arora caches web pages differently.

Arora uses 23% more memory. Though this might not be significant on a regular PC, it is not desirable on a mobile platform.

Vector graphics

For vector graphics editors we tested Inkscape [10] and Karbon [11] against each other. Inkscape is a much used application, written in C++ (using gtkmm). Karbon is a part of KOffice.

These are the results from the memory test, after loading a 4500×2234 pixel , 4.54MB, SVG file. See Table 3.

Inkscape uses a lot more memory than Karbon does, in fact 30% more. When profiling the heap we discovered that Inkscape uses garbage collection to handle dynamically

Application	Effective Resident (KB)	Resident (KB)
Inkscape	99407	105436
Karbon	70692	76844

Table 3: Memory usage, vector graphics applications

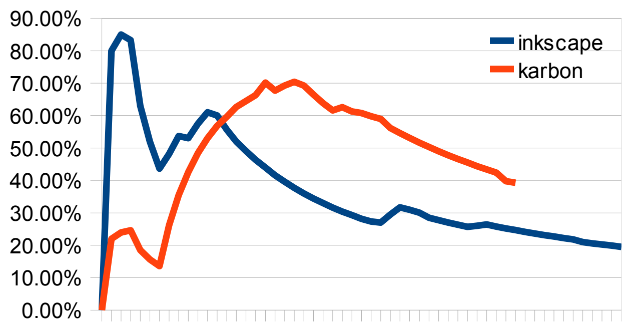

Figure 3: CPU usage, vector graphics applications

allocated memory. As this is known to, in some cases, cause memory fragmentation and therefore lead to excessive memory usage, it could be one explanation for the results.

The CPU test was conducted in this fashion: first we loaded the SVG file, then zoomed to sequentially to 25%, 100%, 800%. See Figure 3.

Inkscape uses more CPU cycles initially, but during zooming Karbon uses the most. Inkscape's interface *seemed* more responsive during the tests.

EXAMPLE APPLICATIONS

Results from our example applications are in Table 4 and Figure 4.

Application	Effective Resident (KB)	Resident (KB)
Qt version	99407	105856
Gtk+ version	30958	86144

Table 4: Memory usage, example applications

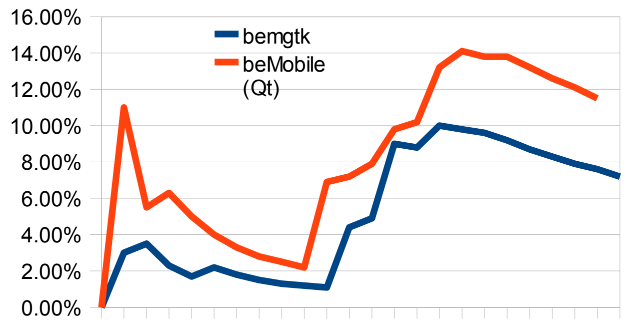

Figure 4: CPU usage, example application

We believe the difference in memory consumption is caused by libWebKit.

Annotation

The results from the example applications like they are presented in this summary, should not be considered accurate due to implementation differences, most notably in the 2D test. It was therefore disabled in the Qt application when the tests was done. Our goal is to improve the implementation and repeat the test once more in near future.

SUMMARY

It is worth to keep in mind that both the QtWebKit and WebKit/GTK+web engines and web browsers we tested are fairly new and still under active development. This is also the case with most of the other applications we tested. The results is therefore only valid until a new version is released, and the tests should be repeated then.

If we had monitored the changes over a longer period of time, the conclusion would probably be that the competition between different applications and their frameworks are important for further improvement in performance on both sides.

The differences caused by the frameworks alone are marginal. We believe that, in general, the implementation done the the programmer plays a larger role.

ACKNOWLEDGEMENTS

This project was commissioned by Qt Software. We would like to express our gratitude to our contact person, Knut Yrvin, for making this project possible and providing useful guidance.

Also we wish to thank our contact at Bergen University College, Bjarte Kileng.

REFERENCES

1.GTK. http://www.gtk.org/

2.Qt. http://www.qtsoftware.com/

3.Exmap introduction.
http://www.berthels.co.uk/exmap/doc.html

4.Valgrind. http://valgrind.org/info/tools.html

5.Evince. http://projects.gnome.org/evince/

6.Okular. http://okular.kde.org/

7.EPDFView. http://trac.emma-soft.com/epdfview/

8.Midori. http://www.twotoasts.de/

9.Arora. http://code.google.com/p/arora/

10.Inkscape. http://www.inkscape.org/

11.Karbon. http://www.koffice.org/karbon/